**Task 1 (1 Hour)**.You are given a table, *Projects*, containing three columns: *Task\_ID*, *Start\_Date* and *End\_Date*. It is guaranteed that the difference between the *End\_Date* and the *Start\_Date* is equal to *1* day for each row in the table.



If the *End\_Date* of the tasks are consecutive, then they are part of the same project. Samantha is interested in finding the total number of different projects completed.

Write a query to output the start and end dates of projects listed by the number of days it took to complete the project in ascending order. If there is more than one project that have the same number of completion days, then order by the start date of the project.

**Sample Input**



**Sample Output**

2015-10-28 2015-10-29

2015-10-30 2015-10-31

2015-10-13 2015-10-15

2015-10-01 2015-10-04

**Task 2 (1 hour).** You are given three tables: *Students*,*Friends*and*Packages.* *Students* contains two columns: *ID* and *Name*. *Friends* contains two columns: *ID* and *Friend\_ID* (*ID* of the ONLY best friend). *Packages* contains two columns: *ID* and *Salary* (offered salary in $ thousands per month).



Write a query to output the names of those students whose best friends got offered a higher salary than them. Names must be ordered by the salary amount offered to the best friends. It is guaranteed that no two students got same salary offer.

**Sample Input**

 

**Sample Output**

Samantha

Julia

Scarlet

**Task 3 (20 mins).** You are given a table, *Functions*, containing two columns: *X*and *Y*.



Two pairs *(X1, Y1)* and *(X2, Y2)* are said to be *symmetric* *pairs* if *X1 = Y2* and *X2 = Y1*.

Write a query to output all such *symmetric* *pairs* in ascending order by the value of *X*.

**Sample Input**



**Sample Output**

20 20

20 21

22 23

**Task 4(2 hour).** Samantha interviews many candidates from different colleges using coding challenges and contests. Write a query to print the *contest\_id*, *hacker\_id*, *name*, and the sums of *total\_submissions*, *total\_accepted\_submissions*, *total\_views*, and *total\_unique\_views* for each contest sorted by *contest\_id*. Exclude the contest from the result if all four sums are .

**Note:** A specific contest can be used to screen candidates at more than one college, but each college only holds  screening contest.

**Input Format**

The following tables hold interview data:

* *Contests:* The *contest\_id* is the id of the contest, *hacker\_id* is the id of the hacker who created the contest, and *name* is the name of the hacker. 
* *Colleges:* The *college\_id* is the id of the college, and *contest\_id* is the id of the contest that Samantha used to screen the candidates. 
* *Challenges:* The *challenge\_id* is the id of the challenge that belongs to one of the contests whose contest\_id Samantha forgot, and *college\_id* is the id of the college where the challenge was given to candidates. 
* *View\_Stats:* The *challenge\_id* is the id of the challenge, *total\_views* is the number of times the challenge was viewed by candidates, and *total\_unique\_views* is the number of times the challenge was viewed by unique candidates. 
* *Submission\_Stats:* The *challenge\_id* is the id of the challenge, *total\_submissions* is the number of submissions for the challenge, and *total\_accepted\_submission* is the number of submissions that achieved full scores. 

**Sample Input**

*Contests* Table:  *Colleges* Table:  *Challenges* Table:  *View\_Stats* Table: *Submission\_Stats* Table: 

**Sample Output**

66406 17973 Rose 111 39 156 56

66556 79153 Angela 0 0 11 10

94828 80275 Frank 150 38 41 15

**Task 5 (45 mins).** Julia conducted a  days of learning SQL contest. The start date of the contest was *March 01, 2016* and the end date was *March 15, 2016*.

Write a query to print total number of unique hackers who made at least  submission each day (starting on the first day of the contest), and find the *hacker\_id* and *name* of the hacker who made maximum number of submissions each day. If more than one such hacker has a maximum number of submissions, print the lowest *hacker\_id*. The query should print this information for each day of the contest, sorted by the date.

**Input Format**

The following tables hold contest data:

* *Hackers:* The *hacker\_id* is the id of the hacker, and *name* is the name of the hacker.
* *Submissions:* The *submission\_date* is the date of the submission, *submission\_id* is the id of the submission, *hacker\_id* is the id of the hacker who made the submission, and *score* is the score of the submission. 

**Sample Input**

For the following sample input, assume that the end date of the contest was *March 06, 2016*.

*Hackers* Table: 

*Submissions* Table: 

**Sample Output**

2016-03-01 4 20703 Angela

2016-03-02 2 79722 Michael

2016-03-03 2 20703 Angela

2016-03-04 2 20703 Angela

2016-03-05 1 36396 Frank

2016-03-06 1 20703 Angela

**Task 6 (20 mins).** Consider P1(a,b) and P2(c,d) to be two points on a *2D* plane.

* happens to equal the minimum value in *Northern Latitude* (*LAT\_N* in **STATION**).
* happens to equal the minimum value in *Western Longitude* (*LONG\_W* in **STATION**).
* happens to equal the maximum value in *Northern Latitude* (*LAT\_N* in **STATION**).
* happens to equal the maximum value in *Western Longitude* (*LONG\_W* in **STATION**).

Query the [Manhattan Distance](https://xlinux.nist.gov/dads/HTML/manhattanDistance.html) between points P1 and P2 and round it to a scale of  decimal places.

**Input Format**

The **STATION** table is described as follows:



where *LAT\_N* is the northern latitude and *LONG\_W* is the western longitude.

**Task 7 (1 hour).** Write a query to print all *prime numbers* less than or equal to 1000. Print your result on a single line, and use the ampersand (&) character as your separator (instead of a space).

For example, the output for all prime numbers <=10 would be:

**Task 8 (2 hours).** [Pivot](https://en.wikipedia.org/wiki/Pivot_table) the *Occupation* column in **OCCUPATIONS** so that each *Name* is sorted alphabetically and displayed underneath its corresponding *Occupation*. The output column headers should be *Doctor*, *Professor*, *Singer*, and *Actor*, respectively.

**Note:** Print **NULL** when there are no more names corresponding to an occupation.

**Input Format**

The **OCCUPATIONS** table is described as follows:



*Occupation* will only contain one of the following values: **Doctor**, **Professor**, **Singer** or **Actor**.

**Sample Input**



**Sample Output**

Jenny Ashley Meera Jane

Samantha Christeen Priya Julia

NULL Ketty NULL Maria

**Task 9 (2 hours).** You are given a table, *BST*, containing two columns: *N*and *P,* where *N* represents the value of a node in *Binary Tree*, and *P* is the parent of *N*.



Write a query to find the node type of *Binary Tree* ordered by the value of the node. Output one of the following for each node:

* *Root*: If node is root node.
* *Leaf*: If node is leaf node.
* *Inner*: If node is neither root nor leaf node.

**Sample Input**



**Sample Output**

1 Leaf

2 Inner

3 Leaf

5 Root

6 Leaf

8 Inner

9 Leaf

**Task 10 (2 hours).** Amber's conglomerate corporation just acquired some new companies. Each of the companies follows this hierarchy: 

Given the table schemas below, write a query to print the *company\_code*, *founder* name, total number of *lead* managers, total number of *senior* managers, total number of *managers*, and total number of *employees*. Order your output by ascending *company\_code*.

**Note:**

* The tables may contain duplicate records.
* The *company\_code* is string, so the sorting should not be **numeric**. For example, if the *company\_codes* are *C\_1*, *C\_2*, and *C\_10*, then the ascending *company\_codes* will be *C\_1*, *C\_10*, and *C\_2*.

**Input Format**

The following tables contain company data:

* *Company:* The *company\_code* is the code of the company and *founder* is the founder of the company. 
* *Lead\_Manager:* The *lead\_manager\_code* is the code of the lead manager, and the *company\_code* is the code of the working company. 
* *Senior\_Manager:* The *senior\_manager\_code* is the code of the senior manager, the *lead\_manager\_code* is the code of its lead manager, and the *company\_code* is the code of the working company. 
* *Manager:* The *manager\_code* is the code of the manager, the *senior\_manager\_code* is the code of its senior manager, the *lead\_manager\_code* is the code of its lead manager, and the *company\_code* is the code of the working company. 
* *Employee:* The *employee\_code* is the code of the employee, the *manager\_code* is the code of its manager, the *senior\_manager\_code* is the code of its senior manager, the *lead\_manager\_code* is the code of its lead manager, and the *company\_code* is the code of the working company. 

**Sample Input**

*Company* Table: 

*Lead\_Manager* Table: *Senior\_Manager* Table: 

*Manager* Table:  *Employee* Table: 

**Sample Output**

C1 Monika 1 2 1 2

C2 Samantha 1 1 2 2

**Task 11 (1 hour).** You are given three tables: *Students*,*Friends*and*Packages.* *Students* contains two columns: *ID* and *Name*. *Friends* contains two columns: *ID* and *Friend\_ID* (*ID* of the ONLY best friend). *Packages* contains two columns: *ID* and *Salary* (offered salary in $ thousands per month).



Write a query to output the names of those students whose best friends got offered a higher salary than them. Names must be ordered by the salary amount offered to the best friends. It is guaranteed that no two students got same salary offer.

**Sample Input**

 

**Sample Output**

Samantha

Julia

Scarlet

**Task 12.** Find top 5 employees according to salary (without order by).

**Task 13.** Swap value of two columns in a table without using third variable or a table.

**Task 14.** Create a user ,create a login for that user provide permissions of DB\_owner to the user.

**Task 15.** Samantha was tasked with calculating the average monthly salaries for all employees in the **EMPLOYEES** table, but did not realize her keyboard's 0  key was broken until after completing the calculation. She wants your help finding the difference between her miscalculation (using salaries with any zeroes removed), and the actual average salary.

Write a query calculating the amount of error (i.e.:  actual – miscalculated average monthly salaries), and round it up to the next integer.

**Task 16.** Copy new data of one table to another( you do not have indicator for new data and old data).

**Task 17 (1 Day)** . Understand the concept of dimension tables in data modelling. Learn the importance and schema structure of **DimDate** table (date dimension table) in modelling and implement a stored procedure code to load 25 years(from today’s date) date data and its computed date fields in date dimension table.

**Task 18 (1 Day).** Learn the importance and schema structure of **DimTime** table (time dimension table) in modelling and implement a stored procedure code to load data and its computed time fields in time dimension table.

**Task 19 (1 Day)**. Understand the concept of Slowing Changing Dimension and its types in Data Modelling. Create a stored procedure to implement SCD-type1 logic for a sample dimension table(take it any table).

**Task 20 (2 Day)**. Create a stored procedure to implement SCD-type2 logic for a sample dimension table(take it any table).

**Task 21 (3 Day)**. Load the following configuration table in your database. Create a dynamic stored procedure that will work over the following configuration table(existing in your DB) to create new tables in database, if the status is ‘New’( along with the primary keys, Clustered indexes, Non- clustered indexes) or add columns for tables with status ‘Old’ with Alter command( which should run only once, even if there are multiple columns listed to be added in your ‘Old’ status tables.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **NewColumn** | **DataTypes** | **Table** | **TableType** | **TableStatus** | **If\_New\_Table\_then\_PrimaryKey** | **Any\_Index(CLI)** | **Any\_Index(NCLI)** |
| Camp\_id | nvarchar(100) | DimCampaign | Dim | New | Yes | Yes |  |
| Total | float | Orders | Fact | Old |  |  | Yes |
| Camp\_type | nvarchar(100) | DimCampaign | Dim | New |  |  |  |
| Department | nvarchar(100) | DimEmployees | Dim | Old |  |  |  |
|  |  |  |  |  |  |  |  |

**Task 22**— you are given Customer\_Details table having 2 columns customer Id as Cust\_id and check-in/checkout date and time as check-in/checkout.

Write a query to output 3 columns as cust\_id, check-in and checkout.

Note: - do not use min, max type of aggregation.

**Input Table:-**

***Cust\_id   Check-in/checkout***

102         21-02-2020 10:32:10

101         22-02-2020 12:45:36

101         20-02-2020 12:45:36

102         22-02-2020 11:12:56

**Output Table:-**

***Cust\_id Check-in Checkout***

101 20-02-2020 12:45:36 22-02-2020 12:45:36

102 21-02-2020 10:32:10 22-02-2020 11:12:56